EE 524 P
Applied High-Performance GPU Computing

LECTURE 8 : Thursday, November 15, 2018

Instructor: Dr. Colin Reinhardt

University of Washington - Professional Masters Program

Autumn 2018

Lecture 8 : Outline

- HW4 Extension: Due 11/18 by 11:59 PM (SUNDAY)
- Final Project Reminders
  - Project Proposals DRAFT: due TODAY
  - Project Proposals FINAL: due WED 11/21 by 11:59 PM
  - Proposal Designs: due 11/29 at 6:00 PM
  - Final Project: due 12/14 by 6:00 PM (<u>no</u> late submissions)
- Final Project Details
- Spatial Domain Image Processing
  - OpenCL Image kernel corrections
  - EX 8a
- Frequency Domain Image Processing: Image Enhancement
  - clFFT
- Parallelizing Partial Differential Equations (PDEs)
- EX 8b

Final Project: Proposals

Project Proposal: 1-2 pages

MUST include:

- Summary of topic/problem to be studied
- List of primary references to be used
- Role of OpenCL in problem solution

If you still don't have a final project idea or are unsure

- email me
- come to Sunday office hour
- look at the course website page: https://canvas.uw.edu/courses/1260593/pages/final-project-ideas

Final Project Details

- Desired Elements to Include
  1. Theoretical analysis of expected performance
     a. Use the theory and metrics we've studied:
        a. Operational Intensity
        b. Speedup, Efficiency (theoretical/asymptotic, measured/empirical), serial fraction
  2. Optimization of NDRange configuration (workgroup, workitem sizes)
  3. Performance profiling
     a. host side timing : Windows Performance Counters (WPC)
     b. device side timing: various events QUEUED-COMPLETE, START-END, etc…
  4. VTune analysis of performance
     a. Execution Unit (EU) utilization, occupancy
     b. Memory subsystem utilization
     c. Bottleneck analysis

Spatial-Domain Image Processing

Image Enhancement with OpenCL

Corrections to the Conv Kernel

```
__kernel void img_conv_filter(__read_only image2d_t inImg, __write_only image2d_t outImg, __constant float* convfilter, uint filtWidth)
{
// use global IDs for output coords
int x = get_global_id(0); // columns
int y = get_global_id(1); // rows

int halfWidth = (int)(filtWidth/2);   // auto-round nearest int

float sum = 0.0f;

int filtIdx = 0;  // filter kernel passed in as linearized buffer array
int2 coords;

for(int i = -halfWidth; i <= halfWidth; i++) // iterate filter rows
{
	coords.y = y + i;
	for(int j = -halfWidth; j <= halfWidth; j++) // iterate filter cols
	{
		coords.x = x + j;
		float pixel = convert_float(read_imageui(inImg, sampler, coords).x); // operate on single component (x = r)
		sum += pixel * convfilter[filtIdx];
		filtIdx++;
	}

}

//write resultant filtered pixel to output image
coords = (int2)(x,y);
write_imageui(outImg, coords, convert_uint4((float4)(sum,sum,sum,1.0f))); // leave a-channel unchanged
}
```

Spatial-domain image processing

- Fix for non-32bit images
  - use stbi_load( ) : force last argument = 4
  - also ensure you allocate your host-side output-image buffer size accordingly

Original

Example Results

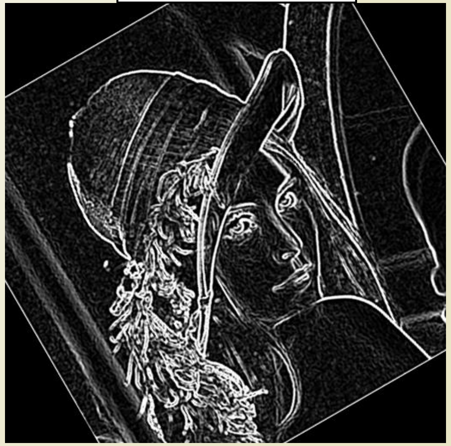
2D Sobel

Basic Laplacian

Composite Laplacian

5x5 Gaussian Blur

Full Chain



In-class Exercise 8a

- Implement the corrections your stencil kernels
- Test the Gauss Blur, Sobel, and Laplacian kernels

- Further experiments
  - Sobel – one dimension only
  - Laplacian – basic versus composite

- Get the full Multiqueue Device-side enqueue processing chain working!
  - Between consecutive kernel device-enqueues…
    - be careful with input-output image object re-ordering
    - be careful with input-output event synchronization
  - Also on host when enqueuing ReadImage… which image is real output??

Frequency-Domain Image Processing

Image Enhancement with OpenCL

clFFT

- Go here: http://clmathlibraries.github.io/clFFT/index.html#Outline
- We'll walk through a code example…

Image Filtering in Frequency Domain

- Image smoothing with Low-pass filters
- Image sharpening with High-pass filters

Parallelizing PDEs

- A VAST topic area
  - Numerical methods for PDEs is a VAST subset
    - Parallel numerical methods for PDEs is a smaller but increasingly important sub-subset
- We wont even begin to scratch the surface in this class
- Will give a brief taste of some approaches, considerations, and techniques
  - some basic techniques are very general and common
- Will gloss over a lot of important details in order to focus on parallelization
  - consistency, stability, convergence
  - underpinnings of various finite methods

PDEs

- Classification:
  - Elliptical
    - Laplace's Equation $\nabla^2 u(x, y, z) = 0$
    - Poisson Equations $\nabla^2 u(x, y, z) = f(x, y, z)$
  - Hyperbolic
    - 2nd-order Wave Equation, $$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$
    - 1D Convection (Advection) Equation $$\frac{\partial u}{\partial t} + c \frac{\partial u}{\partial x} = 0$$
  - Parabolic
    - Heat (Diffusion) Equation $$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

1D Linear Convection Equation

Finite Different Method (FDM)

- Step 1: Discretize the continuous equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \quad \Longrightarrow \quad \frac{u_j^{n+1} - u_j^n}{\Delta t} + c\,\frac{u_{j+1}^{n+1} - u_{j-1}^{n+1}}{2\Delta x} = 0$$

  - Truncation Error (T.E.) $O\left[\Delta t, (\Delta x)^2\right]$

- Step 2 : rewrite with unknowns at time (n+1) on LHS, known u_j^n on RHS

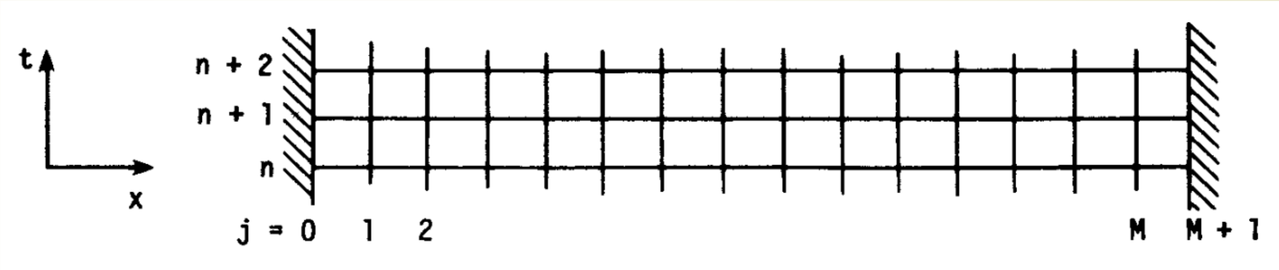
$$a_j{}_{j+1}^{n+1} + d_j u_j^{n+1} + b_j u_{j-1}^{n+1} = C_j$$

  - where $a_j = \frac{v}{2}$, $d_j = 1$, $b_j = \frac{v}{2}$, $C_j = u_j^n$ and the Courant (CFL) number is $v = c\frac{\Delta t}{\Delta x}$

- This is an implicit scheme which is *unconditionally stable* for all time steps

1D Linear Convection Equation : Implicit FD scheme

- Consider computational mesh with M+2 grid points in x direction
  - known initial conditions at *n*=0, boundary conditions u_0^{n+1} and u_{M+1}^{n+1}



- A **tridiagonal** matrix results from solving the system of M linear algebraic equations at each (n+1) time level

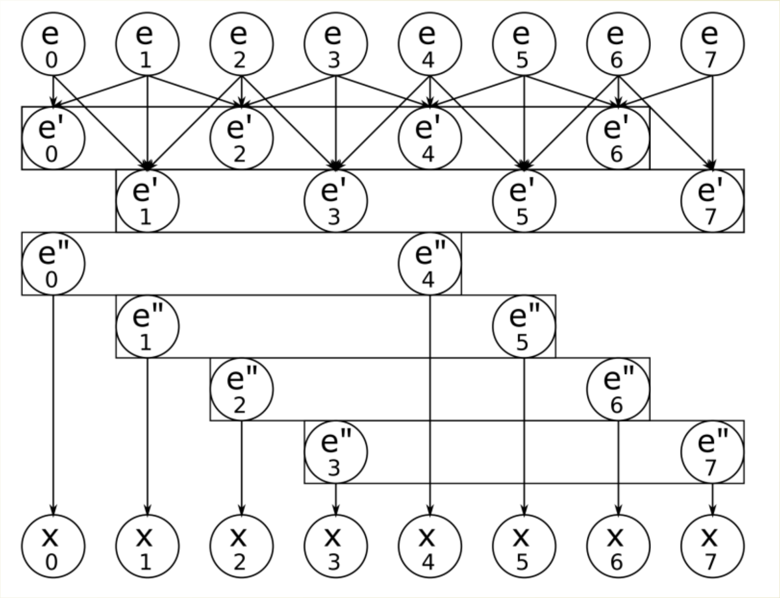
$$
\underbrace{\begin{bmatrix}
d_1 & a_1 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\
b_2 & d_2 & a_2 & & & & & & \cdot \\
0 & b_3 & d_3 & a_3 & & & & & \cdot \\
\cdot & & \ddots & \ddots & \ddots & & & & \cdot \\
\cdot & & & \ddots & \ddots & \ddots & & & \cdot \\
\cdot & & & & \ddots & \ddots & \ddots & & 0 \\
\cdot & & & & & & b_{M-1} & d_{M-1} & a_{M-1} \\
0 & \cdot & \cdot & \cdot & \cdot & & 0 & b_M & d_M
\end{bmatrix}}_{[A]}
\underbrace{\begin{bmatrix}
u_1^{n+1} \\ u_2^{n+1} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ u_{M-1}^{n+1} \\ u_M^{n+1}
\end{bmatrix}}_{[u]}
=
\underbrace{\begin{bmatrix}
C_1 \\ C_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ C_{M-1} \\ C_M
\end{bmatrix}}_{[C]}
$$

Parallel Tridiagonal Solver

- Parallel Cyclic Reduction (PCR) (*Hockney, 1981*)
  - only used forward-reduction phase
  - performed on both *odd* and *even* rows

$$
\begin{array}{c} e_1 \\ e_2 \\ e_3 \\ e_4 \end{array}
\begin{bmatrix} b_1 & c_1 & & \\ a_2 & b_2 & c_2 & \\ & a_3 & b_3 & c_3 \\ & & a_4 & b_4 \end{bmatrix}
\rightarrow
\begin{array}{c} e'_1 \\ e'_2 \\ e'_3 \\ e'_4 \end{array}
\begin{bmatrix} b'_1 & 0 & c'_1 & \\ 0 & b'_2 & 0 & c'_2 \\ a'_3 & 0 & b'_3 & 0 \\ & a'_4 & 0 & b'_4 \end{bmatrix}
\rightarrow
\frac{\begin{bmatrix} b'_1 & c'_1 \\ a'_3 & b'_3 \end{bmatrix}}{\begin{bmatrix} b'_2 & c'_2 \\ a'_4 & b'_4 \end{bmatrix}}
$$

  - Data access pattern of PCR algorithm
    - multi-level tree decomposition
    - obtain independent tasks



  - Computational complexity: $O(n \log n)$
  - Required number of elimination steps: $\log n + 1$

2D Shallow Water Equations (SWEs)

$$\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Matrix form:

$$Q_t + F(Q)_x + G(Q)_y = 0$$

Q is vector of conserved variables
F, *G* are flux functions
h is water depth
hu, *hv* are momenta components along x, y directions
g is gravitational acceleration

Finite Volume Method (FVM)

- Step 1: Discretize the continuous equations : *Lax-Friedrichs explicit scheme*

$$Q_{ij}^{n+1} = \frac{1}{4}\left(Q_{i,j+1}^n + Q_{i,j-1}^n + Q_{i+1,j}^n + Q_{i-1,j}^n\right) - \frac{\Delta t}{2\Delta x}\left[F\left(Q_{i+1,j}^n\right) - F\left(Q_{i-1,j}^n\right)\right] - \frac{\Delta t}{2\Delta y}\left[G\left(Q_{i,j+1}^n\right) - G\left(Q_{i,j-1}^n\right)\right]$$

  - This is a stencil form!
    - four nearest spatial neighbors
- Time-marching method

Parallel 2D SWEs

$$Q_{ij}^{n+1} = \frac{1}{4}\left(Q_{i,j+1}^{n} + Q_{i,j-1}^{n} + Q_{i+1,j}^{n} + Q_{i-1,j}^{n}\right) - \frac{\Delta t}{2\Delta x}\left[F\left(Q_{i+1,j}^{n}\right) - F\left(Q_{i-1,j}^{n}\right)\right] - \frac{\Delta t}{2\Delta y}\left[G\left(Q_{i,j+1}^{n}\right) - G\left(Q_{i,j-1}^{n}\right)\right]$$

- Serial Algorithm (for computing h^{n+1})

```
for (int j=1; j<ny-1; ++j) {
   for (int i=1; i<nx-1; ++i) {
      int n=(j+1)*nx+i, s=(j-1)*nx+i, e=j*nx+i+1, w=j*nx+i-1;
      h_new[j*nx+i] = 0.25*(h[n]+h[s]+h[e]+h[w])
         - 0.5*dt/dx*(hu[e]-hu[w]) - 0.5*dt/dy*(hv[n]-hv[s]);
   }
}
```

- Simple OpenCL Kernel

```
__kernel void SWELaxFriedrichs(global float* h_new, global float* h, global float* hu,
                               global float* hv, int nx, int ny)
{
     int i = get_global_id(0);
     int j = get_global_id(1);
     if (i > 0 && i < nx-1 && j > 0 && j < ny-1) {
        int n=(j+1)*nx+i, s=(j-1)*nx+i, e=j*nx+(i+1), w=j*nx+i-1;
        h_new[j*nx+i] = 0.25*(h[n]+h[s]+h[e]+h[w])
           - 0.5*dt/dx*(hu[e]-hu[w]) - 0.5*dt/dy*(hv[n]-hv[s]);
        }
}
```

In-class Exercise 8b

- clFFT
- See procedures on class website /InclassExercises/Ex7